REMARKS

I. CLAIM REJECTIONS UNDER §112

Claims 17 was rejected under §112, first paragraph as allegedly failing to comply with the enablement requirement.

Applicants respectfully disagree since the input and output channels and room builder objects are described in the specification in a manner that could easily be understood by a person having ordinary skill in the art.

The TCL Command Interpreter includes a number of channels. An Input channel is used to take input from the user, and an Output channel is used to display output to the user. When the TCL interpreter is created its Input/Output channels are connected to the main application. After this point, commands entered by the user from the input window are passed to the TCL interpreter through the Input channel, TCL outputs are displayed on the corresponding graphic objects determined by the application.

The last paragraph appearing on page 26 (see also 1st para. on page 28) describes that the command processor creates a TCL interpreter object, connects input/output channels, and creates room builder objects. It is obvious from the specification that user commands are input to the command interpreter and that resulting outputs from the command interpreter are output for display on the corresponding graphic objects.

Room Builder objects are described in more detail on page 28, for example:

> The command processor 410 creates a TCL interpreter object (step 502), and connects input and output channels (step 504). Then, the command processor 410 creates <u>room builder objects</u> (step 506). In this step, base class object constructors add various <u>builder object command names</u> to the TCL namespace, and destructors remove commands and object command names from the TCL namespace.
>
> The command processor 410 loads user specified TCL command configuration scripts (step 508), either from a file or from direct user input. In general, <u>the user specified TCL command configuration scripts may be understood to be user specified room commands</u>. Objects created by the user in step 506 process these commands. <u>Once rooms and room windows are created</u>, menus, menu items, buttons, accelerator keys, and the like can be added to

specific room windows. The addition of such elements within the room windows also requires that the user specify the specific functionality to which the various objects correspond.

Finally, the command processor 410 displays the windows (step 510), which were created by the user specified TCL commands, and enters the event loop.

In any case, claim 17 is cancelled to simplify the issues in this application. Applicants reserve the right to pursue claim 17 in one or more continuation applications.

II. CLAIM REJECTIONS UNDER §102

Claims 1-4, 7-13 and 18-20 were rejected under 35 U.S.C. §102 (b) as allegedly being anticipated by Nahaboo et al. U.S. Patent No. 5,974,253.

A. **General Discussion**

The following general discussion is not intended to limit any particular claim or claim element, but is provided as a general background.

Nahaboo et al. describe a system that allows the user to design and configure a user interface using a program in edit mode and then test it in an execution mode. This process is manual and takes a relatively long time (several minutes to several hours) to complete.

An example of the command processor described in the present application eliminates the need for a separate edit or configuration tool. The command processor allows the user to configure the graphical user interface using TCL configuration commands, for example, wherein the commands may be built-in system commands, existing TCL scripts or TCL function/scripts created by the user at run time. The command processor provides the ability to change the interface an unlimited number of times during the same execution session, and these changes can be made in a few milliseconds to a few seconds, for example.

In one example, assume "User A" has a session up and is working on a design, wherein User A has configured the interface to his or her preferences. If "User B" decides to use the same session he or she can change the GUI configuration by running his or her own configuration script. This change may take only a few milliseconds. At a later time, User A may continue work on the same session using his or her own configuration by just running his or her own configuration script to restored GUI to User A's preferences.

Nahaboo et al. contains an editor (described in Colum2, Line 49) that allows the user to arrange or modify a preexisting set of graphical objects using a manual interactive method. Using this technique, the time to change the user interface configuration may range form a few minutes to a few hours.

Nahaboo et al. describe in column 3, lines 53-56,

> "In order to function, this interactive description interface (10) must be associated with a library of interactive command objects (called command object toolbox "X/MOTIF" (20) in the UNIX environment) and a graphical object toolbox that includes a library of graphical objects "GO" (21)." (Emphasis added).

In other words, the commands and objects are pre-existing and hard-coded. Column 9, lines 36-42 state,

> "Selecting the 'geometry' menu, using button 3111, will activate an editing function that allows the user to display selection and modification attributes of a widget's geometry. The "behavior" menu, activated using button 3112, allows the user to activate an editing function that will display the attribute that defines a widget's behavior and what its behavior will be when using the attributes selected." (Emphasis added).

This manual, on-screen selection and modification process can therefore be very time-consuming and not easily changed from one session to the next or between users. A new user to the same session would therefore need to repeat each button selection and editing operation described above for each widget, etc. the is desired to be changed.

Nahaboo et al. describe that, "We have used the word "script" to designate the code that describes the response to the set of callbacks" (col. 11, lines 55-56) and that, "The callback procedure section is simply a text area in which the interactive "widget" callback procedures can be edited" (col. 9, lines 53-55).

An example of the command processor described in the present disclosure eliminates this editor altogether and the user does not need an additional tool to create or modify the interface. The interface is created or modified through TCL commands, which may be completely different from one invocation to the next and the command themselves may not have existed before the invocation of the tools. Put another way, new commands can be created by the user on the fly and then used to configure the GUI based on the new command. Using this technique, the user interface can be changed from one user configuration to the next using each user's interface configuration file, and this change may take only a few milliseconds to a few seconds.

An example of the command processor eliminates the need for all functions described in Nahaboo in column 2, line 49 to column 3, line 15.

Further, Nahaboo et al. have a table (described in column 4, line 63 and in Fig 1B) that stores the association between a graphical object a selector and a call back in the interpreter. These are fixed and predetermined when the tool is invoked.

In an example of the command processor described in the present disclosure, there is no required predetermined table or mechanism that stores this association. This allows the user to add an unlimited number of commands and tie them to graphical objects at run time. The list of methods described in Nahaboo must exist (compiled into the system) before the user can use it. In an example described in the present disclosure, there is no hard-coded list of methods to be used when the system in compiled. All the methods are defined at run time, thus affording the user an unlimited number of methods to be tied to graphical objects.

Column 9, line 36 of Nahaboo et al. describes a fixed relationship between graphical objects (in this case a menu), and selecting them will activate a function that may be changed by the user using an editor. This function determines the action performed when the user selects an object. This function exists prior to the assignment and cannot be selected in less than a number of minutes.

B. **Claim 1**

Looking at claim 1, Nahaboo et al. do not anticipate:

> a command interpreter, which loads one or more configuration commands into the command processor from a user specified command configuration script.

Nahaboo et al. also do not disclose,

> a command interpreter, which loads one or more configuration commands into the command processor from a command line in which the one or more configuration commands are entered by the user.

Nor does Nahaboo et al. interpret such configuration commands or modify a graphical user interface according to such interpreted configuration commands.

Rather, Nahaboo et al. uses an editor that requires the user to arrange or modify a preexisting set of graphical objects to build a user interface using a manual interactive method. Configuration commands are not loaded from a user-specified command configuration script or loaded from a command line when entered by the user.

C. **Independent Claim 8**

Looking at claim 8, the Examiner states that since Nahaboo et al. mention an interpreted language (Abstract, lines 1-2), Nahaboo et al. sufficiently disclose the elements of claim 8. However, Nahaboo et al. do not utilize commands in the manner recited in claim 8.

Nahaboo et al. do not build graphical objects or assign functionality to built graphical objects by loading one or more TCL configuration commands into a command processor from a user specified TCL command configuration script comprising the one or more TCL commands.

Nor do Nahaboo et al. build graphical objects or assign functionality by loading one or more TCL configuration commands into a command processor from a command line in which the one or more TCL configuration commands are entered by the user.

Further, Nahaboo et al. do not display a user-interactive window containing graphical objects according to such TCL commands.

D. **Independent Claim 19**

Claim 19 is similar to claim 8 and adds that the graphical user interface has no hard coded objects. Rather, the graphical user interface is assembled based on the interpreted configuration commands from the user. All objects within the graphical user interface are user defined through the one or more configuration commands.

These elements are not anticipated by Nahaboo et al.

Since Nahaboo et al. do not anticipate the elements of independent claims 1, 8 and 19, Applicants respectfully request that the rejections of these claims and their respective dependent claims under §102(b) be withdrawn.

III. <u>CLAIM REJECTIONS UNDER §103(a)</u>

Claims 5-6, 14-17, and 21-24 were rejected as allegedly being anticipated by Nahaboo et al. in view of Dangelo U.S. Patent No. 5,493,508.

Dangelo was cited merely for disclosing a suite of integrated circuit design tools.

Neither Nahaboo et al. nor Dangelo disclose the elements discussed above with respect to independent claims 1, 8 and 19. Thus, even if Nahaboo et al. were modified as suggested in the Office Action, the resulting modification would still lack these elements.

These elements would not be obvious to a person of ordinary skill in the art since Nahaboo et al. disclose such a different method for altering an interface. Nahaboo et al. require selection from predefined objects and functions, such as from a table of functions or attributes.

Similarly, independent claim 24 includes a command processor having a graphical user interface and a command interpreter for interpreting user commands. The graphical user interface is <u>specified entirely</u> by a user <u>through one or more configuration commands loaded into the command processor</u> at run time of the command processor and interpreted by the command interpreter. One or more design tools corresponding to processes within an integrated circuit design process operate under control of the command processor and within the graphical user interface.

Nahaboo et al. do not teach or suggest such a command processor, and certainly do not make obvious a command processor, where the graphical user interface is specified entirely by a user through one or more configuration commands loaded into the command processor at run time of the command processor and interpreted by a command interpreter.

Applicants therefore respectfully request that the claim rejections under §103(a) based on Nahaboo et al. and Dangelo be withdrawn.

The Director is authorized to charge any fee deficiency required by this paper or credit any overpayment to Deposit Account No. 12-2252.

Respectfully submitted,

WESTMAN, CHAMPLIN & KELLY, P.A.

By: /David D. Brush/

David D. Brush, Reg. No. 34,557
900 Second Avenue South, Suite 1400
Minneapolis, Minnesota 55402-3319
Phone: (612) 334-3222 Fax: (612) 334-3312